GTSuite Designer

Technical Documentation

Delphi Metadata Designer & Application Builder

GTsoftware di Giancarlo Thiella

Generated: November 20, 2025

Table of Contents

1. Executive Summary

GTSuite Designer is a sophisticated Delphi application that serves as the visual development environment for creating metadata-driven applications. Built using Delphi and the A1Suite component framework, it provides a complete IDE for designing application interfaces, defining business logic, and generating metadata that powers the entire GTSuite ecosystem.

Key Characteristics

- **Meta-Circular Design:** The designer itself is built using metadata principles
- **Visual IDE:** Complete visual development environment for metadata creation
- **A1Suite Powered:** Built on reusable, metadata-driven Delphi components
- **SQLite Export:** Generates SQLite databases containing complete application metadata
- **Multi-Project:** Single designer manages multiple application projects
- **Template System:** GTS_Template.db provides starting point for new projects
- **Comprehensive Tools:** Page designer, menu designer, SQL repository, report builder
- **Direct Upload:** Built-in uploader to Node.js server for metadata deployment
- **Database Agnostic:** Supports Oracle, SQL Server, PostgreSQL, MongoDB
- **Validation:** Built-in metadata validation before export

2. Meta-Circular Architecture

The Recursive Philosophy

CRITICAL CONCEPT: The GTSuite Designer embodies a meta-circular design philosophy where the tool used to create metadata-driven applications is itself built using metadata. This recursive architecture provides both consistency and serves as a living example of the framework's capabilities.

Three Database Layers

Database	Purpose	Usage
A1SPrj_GTS.db	Designer's own metadata	Defines GTS Designer's interface
GTS_Template.db	Empty project template	Starting point for new projects
GTS_GTSW.db	Project metadata (example)	Admin module metadata for Node.js

The Complete Flow

Level 1 - Designer Bootstrap: A1SPrj_GTS.db contains metadata that defines the GTS Designer application itself

Level 2 - Designer Usage: Designer loads A1SPrj_GTS.db and renders its own interface using A1Suite components

Level 3 - Project Creation: Designer creates/edits project databases (GTS_GTSW.db, GTS_GTR.db, etc.)

Level 4 - Export: Project databases exported as SQLite files

Level 5 - Upload: SQLite files uploaded to Node.js server

Level 6 - Import: Node.js imports SQLite into MongoDB

Level 7 - Consumption: Angular/Ionic client renders UI from MongoDB metadata

Why Meta-Circular?

- **Dogfooding:** The designer uses the same components and patterns it generates
- **Living Example:** The designer's source code serves as reference implementation
- **Consistency:** All applications share the same architectural foundation
- **Validation:** If the designer works, the generated applications will work

- **Evolution:** Improvements to the framework immediately benefit the designer itself

3. A1Suite Packages (Core Components)

The A1Suite packages (PackagesSources) contain the core Delphi components that power both the GTS Designer and all generated applications. These reusable components implement the metadata-driven paradigm and provide a consistent interface across all applications.

Core Package Files

File	Purpose
A1Common.pas	Core data structures, base classes, and common utilities
A1DBconn.pas	Database connection management (Oracle, MSSQL, PostgreSQL)
A1FormDef.pas	Form and page definition classes
A1RepConn.pas	Report connection and execution
A1SQLDBClass.pas	SQL database abstraction layer
A1UserData.pas	User authentication and session management
A1Startup.pas	Application initialization and licensing
A1LicenseJWT.pas	JWT-based license validation

Key Classes & Components

- **TA1CommonData:** Central data repository, metadata storage, application state
- **TA1Connections:** Database connection definitions and pooling
- **TA1FormDef:** Form definitions, field layouts, validation rules
- **TA1SQLDBClass:** Abstract SQL execution layer supporting multiple databases
- **TA1UserData:** User authentication, roles, permissions, session management
- **TA1EditMask:** Input mask definitions for field formatting
- **TA1StyleParams:** Visual styling and appearance settings
- **TA1SharedSessions:** Multi-user session coordination

Component Philosophy

The A1Suite components follow a data-driven design where configuration is stored as metadata rather than code. Components read their configuration at runtime, allowing the same compiled binary to behave differently based on the loaded metadata. This principle extends from simple field formatting to complex business logic execution.

4. GTS Designer Application

The GTS Designer application (GTS_Sources) is the main visual development environment. It provides a comprehensive IDE for creating, editing, and managing application metadata across multiple projects.

Application Modules

Module	File	Purpose
Main Form	GTS_MAINFORM	Main application window and project switcher
Project Selector	GTS_PRJ_SELECT	Project selection and management
Pages Designer	GTS_PAGES	Form and page metadata designer
Menu Designer	GTS_MENU_DESIGNER	Hierarchical menu structure designer
SQL Repository	GTS_SQL_REPOSITORY	SQL query and procedure management
Connections	GTS_CONNECTIONS	Database connection configuration
Setup	GTS_SETUP	Global settings and configuration
Messages	GTS_MESSAGES	User message templates
Icons Library	GTS_ICONS	Icon management
Report Builder	GTS_RPT_BUILDER	Report definition tool
Metadata Uploader	GTS_METADATA_UPLOADER	Node.js upload utility

Startup & Licensing

The application uses JWT-based licensing with expiration tracking. On startup, it validates the license, loads the project repository, and presents the project selector. The license path and validation are handled by A1Startup.pas using JWT tokens.

```
// GTS.dpr startup
pA1Common := TA1CommonData.Create;
bRun := A1LoadRepository(pA1Common);

if bRun then
  Application.CreateForm(TfrmGTS, frmGTS);
  Application.Run;
```

5. SQLite Database Schema

The SQLite database schema defines the structure for storing all application metadata. Each table represents a different aspect of the application: forms, grids, actions, menus, reports, etc. The schema is identical across all projects, starting from GTS_Template.db.

Core Metadata Tables

Table	Purpose
GtsForms	Page/form definitions with URLs and initialization actions
GtsPageFields	Field definitions with labels, validation, formatting
GtsDataSets	Data source definitions linking to SQL queries
GtsGrids	Grid configurations with columns and behaviors
GtsColumns	Grid column definitions with formatting
GtsTabs	Tab interface definitions
GtsToolbar	Toolbar configurations
GtsToolbarItems	Individual toolbar buttons and actions
GtsActions	Action definitions (event handlers)
GtsActionsHdr	Action group headers
GtsViews	View configurations (list, detail, etc.)
GtsMessages	User-facing messages and dialogs

SQL & Data Access Tables

- **GtsSQL:** SQL query definitions (SELECT, INSERT, UPDATE, DELETE, PROCEDURE)
- **GtsSQLVariables:** Query parameter definitions
- **GtsSQLAllColumns:** Column metadata from SQL queries
- **GtsSQLPKColumns:** Primary key column identification
- **GtsMDBCollections:** MongoDB collection definitions
- **GtsMDBOperations:** MongoDB operation definitions
- **GtsMDBFields:** MongoDB field mappings

Configuration Tables

- **GtsSetup:** Global project configuration
- **GtsConnections:** Database connection definitions
- **GtsMailServices:** SMTP email service configuration
- **GtsDataSchemas:** Database schema definitions
- **GtsScheduledTask:** Cron job definitions

Security & Access Control

- **GtsUsers:** User account definitions
- **GtsRoles:** Role definitions
- **GtsAuthProfileHdr:** Authentication profile headers
- **GtsAuthProfileRoles:** Profile to role mappings
- **GtsPasswPolicies:** Password policy definitions
- **GtsPageObjGranted:** Object-level permission grants
- **GtsMenuRoles:** Menu item role restrictions

6. Metadata Creation Workflow

Creating a New Project

Step 1 - Template Copy: Copy GTS_Template.db to GTS_NEWPROJECT.db

Step 2 - Open Designer: Launch GTS Designer, select new project

Step 3 - Configure: Set project ID, description, connections in GtsSetup

Step 4 - Database Setup: Define database connections in GTS_CONNECTIONS

Step 5 - Menu Structure: Design menu hierarchy in GTS_MENU_DESIGNER

Step 6 - SQL Repository: Create SQL queries and procedures

Step 7 - Page Design: Create pages with forms and grids

Step 8 - Actions: Define user actions and business logic

Step 9 - Validation: Test and validate metadata

Step 10 - Export: Save and prepare for upload

Development Best Practices

- **Start Simple:** Begin with basic CRUD pages, add complexity incrementally
- **Reuse SQL:** Create reusable SQL queries in repository, reference by ID
- **Naming Convention:** Use consistent naming for objects (ds*, frm*, grd*)
- **Test Incrementally:** Upload and test pages frequently during development
- **Document Actions:** Add clear descriptions to action definitions
- **Version Control:** Keep SQLite database files in version control
- **Backup Regularly:** SQLite files are the source of truth - backup often
- **Use Templates:** Create page templates for common patterns

7. Project Templates

Template Database System

The GTSuite uses a template-based approach to project creation. GTS_Template.db serves as the foundation for all new projects, providing an empty but properly structured database schema ready for metadata population.

Available Templates

Database	Purpose	Usage
GTS_Template.db	Empty project template	Starting point for any new project
GTS_GTSW.db	Admin module (example)	Reference for admin functionality
A1SPrj_GTS.db	Designer metadata	Reference for complex applications

Template Creation Process

Copy template:

```
copy GTS_Template.db GTS_MYAPP.db
```

Set project ID:

```
Update GtsSetup: PRJ_ID = 'MYAPP'
```

Configure connections:

```
Add entries to GtsConnections
```

Open in designer:

```
GTS Designer → Open GTS_MYAPP.db
```

8. Forms & Pages Designer

The Pages Designer (GTS_PAGES) is the core module for creating application pages. It provides a visual interface for defining forms, fields, validation rules, and page structure.

Page Components

- **Form Definition:** Page ID, URL, title, initialization action
- **Field Groups:** Logical grouping of fields (can be popup forms)
- **Page Fields:** Individual field definitions with data binding
- **Data Sets:** Data source configurations linked to SQL queries
- **Tabs:** Multi-tab layouts for complex pages
- **Toolbars:** Action buttons and command bars
- **Views:** Different display modes (list, detail, edit)
- **Conditional Rules:** Field visibility and validation based on conditions
- **Messages:** User-facing notifications and confirmations

Field Types & Formatting

- Text inputs with masks and validation
- Numeric inputs with precision and formatting
- Date/Time pickers with format strings
- Lookup fields with searchable dropdowns
- Checkboxes and boolean fields
- File upload fields
- Rich text editors
- Calculated/derived fields

Page Creation Workflow

1. Create form entry in GtsForms (FORM_ID, URL, title)
2. Define data sets in GtsDataSets (link to SQL queries)

3. Create field groups in GtsFldGroups
4. Add fields in GtsPageFields (bind to data set columns)
5. Configure tabs if needed in GtsTabs
6. Add toolbar with action buttons in GtsToolbar
7. Define views (list, detail) in GtsViews
8. Set up conditional rules in GtsExecCondRules
9. Add user messages in GtsMessages

9. Grids & Data Components

Grid components provide tabular data display with sorting, filtering, editing, and selection capabilities. Grids are configured through metadata defining columns, data sources, and behaviors.

Grid Configuration

- **Data Source:** Link to GtsDataSets defining the query
- **Columns:** Column definitions in GtsColumns with formatting
- **Bands:** Column grouping headers in GtsGridBands
- **Sorting:** Default sort columns and directions
- **Filtering:** Built-in filter capabilities
- **Selection:** Single or multi-row selection
- **Editing:** Inline editing with validation
- **Paging:** Server-side or client-side pagination
- **Images:** Conditional image columns in GtsColumnsImg

Data Sets & SQL Binding

Data sets (GtsDataSets) define the connection between UI components and database queries. Each data set:

- Links to a SQL query in GtsSQL (SELECT query)
- Optionally links to INSERT, UPDATE, DELETE queries for CRUD operations
- Supports master-detail relationships between data sets
- Defines data adapter for MongoDB operations (alternative to SQL)
- Manages query parameters and variable binding

10. Actions & Business Logic

Actions represent user interactions and business logic execution. They are defined in metadata and executed by the client framework, enabling complex workflows without client-side code changes.

Action Types

- **EXEC_QUERY:** Execute a SQL query or stored procedure
- **OPEN_FORM:** Navigate to another page/form
- **SAVE_DATA:** Save form data (INSERT or UPDATE)
- **DELETE_DATA:** Delete selected records
- **REFRESH_DATA:** Reload data set
- **SHOW_MESSAGE:** Display message to user
- **EXPORT_EXCEL:** Export grid data to Excel
- **GENERATE_REPORT:** Execute report generation
- **CUSTOM:** Execute custom client-side logic
- **SEQUENCE:** Execute multiple actions in sequence

Conditional Execution

Actions can have execution conditions defined in GtsExecCondRules. Conditions evaluate:

- Field value comparisons (equals, not equals, greater than, etc.)
- Record selection state (has selection, no selection)
- Data set state (has changes, is empty)
- Combined conditions (AND, OR logic)
- Field visibility and enabled state

Action Definition

```
Action metadata includes:
- Action name and description
- Action type (from GtsActionTypes)
- Target SQL query or form
- Parameters and variable bindings
- Success/failure message IDs
- Conditional execution rules
- Confirmation requirements
```

11. Menu Designer

The Menu Designer (GTS_MENU_DESIGNER) creates hierarchical navigation structures. Menus are tree-based with support for multi-level nesting, icons, and role-based visibility.

Menu Structure

- **GtsMenuTree:** Tree structure with parent-child relationships
- **GtsMenuItems:** Leaf nodes linking to forms/pages
- **GtsMenuRoles:** Role-based menu item visibility
- **Icons:** Icon assignments for visual identification
- **Translations:** Multi-language support via TXT_ID

Menu Creation Process

1. Create root menu items in GtsMenuTree (PARENT_ID = 0)
2. Add child items with appropriate PARENT_ID
3. Link leaf items to forms via GtsMenuItems (FORM_ID)
4. Assign icons from icon library
5. Configure role restrictions in GtsMenuRoles
6. Add translations for multi-language support
7. Test menu navigation in client application

Role-Based Menu Filtering

Menu items can be restricted to specific roles through GtsMenuRoles. When a user logs in, the client application requests the menu structure filtered by the user's roles. Items without role restrictions are visible to all users.

12. Export & Upload Pipeline

Export Process

Once metadata design is complete, the SQLite database file must be uploaded to the Node.js server for import into MongoDB. The GTS Designer includes a built-in uploader (GTS_METADATA_UPLOADER) for this purpose.

Step 1 - Validate: Verify metadata consistency and completeness

Step 2 - Save: Ensure all changes are saved to SQLite database

Step 3 - Prepare: Check database file location and accessibility

Step 4 - Configure Upload: Set Node.js server URL and authentication

Step 5 - Select Data: Choose what to upload (setup, pages, full database)

Step 6 - Upload: Transfer SQLite file to server

Step 7 - Import: Server imports SQLite into MongoDB

Step 8 - Verify: Check import logs for errors

Step 9 - Test: Access client application to verify changes

Upload API Integration

The uploader communicates with the Node.js server via REST API:

- **/api/sqlite/loadSetup:** Upload setup configuration only
- **/api/sqlite/loadPage2:** Upload specific page metadata
- **/api/sqlite/openConn:** Establish SQLite connection on server
- **/api/sqlite/closeConn:** Close SQLite connection
- **/api/sqlite/removePageData:** Clear existing page before re-import

Import Verification

After upload, check the GtsSqliteLog collection in MongoDB to verify successful import. The log contains:

- Import timestamp and duration
- Number of records imported per table
- Any errors or warnings encountered

• Validation results

• Performance metrics

13. Integration with Node.js/Angular

The Complete Pipeline

The GTSuite represents a complete metadata-driven application development pipeline spanning three major components:

Component	Role	Technology
Designer	Metadata creation	Delphi + SQLite
Server	Metadata storage & API	Node.js + MongoDB
Client	Metadata consumption	Angular/Ionic

Data Flow

1. **Design:** Create metadata in GTS Designer (Delphi)
2. **Export:** SQLite database file (GTS_PROJECT.db)
3. **Upload:** Transfer SQLite to Node.js server
4. **Import:** Node.js imports SQLite → MongoDB
5. **Transform:** Data normalized and optimized in MongoDB
6. **API:** Client requests metadata via REST API
7. **Render:** Angular/Ionic renders UI dynamically
8. **Execute:** User actions trigger database operations
9. **Update:** Changes flow back to databases via server

Benefits of the Integrated System

• **Separation of Concerns:** Design, deployment, and consumption are independent

• **Version Control:** SQLite files can be versioned alongside code

• **Rapid Iteration:** UI changes without code deployment

• **Consistency:** Same metadata drives all components

• **Scalability:** Designer doesn't need to scale with users

• **Offline Design:** Metadata can be created without server access

- **Testing:** Changes can be tested before deployment
- **Rollback:** Easy to revert to previous metadata versions

Conclusion

The GTSuite Designer represents the foundational tool in a sophisticated metadata-driven development ecosystem. By providing a visual interface for application design while maintaining its own metadata-driven architecture, it demonstrates the power and flexibility of the GTSuite approach. The meta-circular design philosophy - where the tool itself embodies the principles it enables - ensures consistency, validates the architecture, and provides a living reference implementation for developers. Together with the Node.js server and Angular/Ionic client, the GTSuite Designer completes a comprehensive platform for rapid application development with unprecedented flexibility and maintainability. For additional support or inquiries, please contact GTsoftware di Giancarlo Thiella.